Working paper

2020-06

Statistics and Econometrics
ISSN 2387-0303

A Solution Method for the Shared Resource Constrained Multi-Shortest Path Problem

David García-Heredia, Elisenda Molina,

Manuel Laguna, Antonio Alonso-Ayuso

uc3m | Universidad Carlos III de Madrid

A Solution Method for the Shared Resource Constrained Multi-Shortest Path Problem*

David García-Heredia[a,*], Elisenda Molina[a], Manuel Laguna[b], Antonio Alonso-Ayuso[c]

[a] *Departamento de Estadística, Universidad Carlos III de Madrid, Getafe (Madrid), Spain*
[b] *Leeds School of Business, University of Colorado Boulder, USA*
[c] *Área de Estadística e Investigación Operativa, Universidad Rey Juan Carlos, Móstoles (Madrid), Spain*

Abstract

We tackle the problem of finding, for each network within a collection, the shortest path between two given nodes, while not exceeding the limits of a set of shared resources. We present an integer programming (IP) formulation of this problem and propose a parallelizable matheuristic consisting of three phases: 1) generation of feasible solutions, 2) combination of solutions, and 3) solution improvement. We show that the shortest paths found with our procedure correspond to the solution of the Resource-Constrained Multi-Project Scheduling Problem (RCMPSP) and that a particular case of the RCMPSP occurs in Air Traffic Flow Management (ATFM). Our computational results include finding optimal solutions to small and medium-size ATFM instances by applying Gurobi to the IP formulation. We use those solutions to assess the quality of the output produced by our proposed matheuristic. For the largest instances, which correspond to actual flight plans in ATFM, exact methods fail and we assess the quality of our solutions by means of Lagrangian bounds. Computational results suggest that the proposed procedure is an effective approach to the family of shortest path problems that we discuss here.

Keywords: Matheuristics, Shortest Path, Resource-Constrained Multi-Project Scheduling Problem, Air Traffic Flow Management.

2010 MSC: 00-01, 99-00

*This research was partially funded by projects MTM2015-63710-P and RTI2018-094269-B-I00 (A. Alonso-Ayuso and D. García-Heredia) from the Government of Spain

*Corresponding author.

Email addresses: dgheredi@est-econ.uc3m.es (David García-Heredia), emolina@est-econ.uc3m.es (Elisenda Molina), laguna@colorado.edu (Manuel Laguna), antonio.alonso@urjc.es (Antonio Alonso-Ayuso)

1. Introduction

Consider a directed network $\mathcal{G} = (\mathcal{V}, \mathcal{A}, \mathcal{C})$, where $\mathcal{V}$, is the set of vertices (or nodes), $\mathcal{A}$ the set of arcs, and $\mathcal{C}$ the set costs such that, for each arc $a = (i, j) \in \mathcal{A}$, $c_a \in \mathcal{C}$ represents the cost of using arc a to go from node i to node j. The shortest path between an origin (or source) node $o \in \mathcal{V}$ and a destination (or sink) node $d \in \mathcal{V}$ is the one that minimizes the arithmetic sum of the costs associated with the arcs in the path. Finding this minimum-cost path is known as the Shortest Path Problem (SPP) in $\mathcal{G}$ for nodes $o, d \in \mathcal{V}$.

The SPP can be formulated as an Integer Program (IP) by defining, for each arc $a \in \mathcal{A}$, a binary variable x_a that equals 1 if the associated arc is in the shortest path and 0 otherwise. By defining $\Lambda^+(i)$ and $\Lambda^-(i)$ as the set of outgoing and incoming arcs of node $i \in \mathcal{V}$, respectively, the IP formulation of the SPP can be stated as follows:

$$\min \sum_{a \in \mathcal{A}} c_a x_a, \tag{1}$$

subject to:

$$\sum_{a \in \Lambda^+(i)} x_a - \sum_{a \in \Lambda^-(i)} x_a = \begin{cases} 1, & \text{if } i = o, \\ -1, & \text{if } i = d, \\ 0, & \text{otherwise.} \end{cases} \qquad \forall i \in \mathcal{V} \tag{2}$$

$$x_a \in \{0, 1\}, \quad \forall a \in \mathcal{A}. \tag{3}$$

In this formulation, (1) establishes the objective of finding the path with the minimum cost, while (2) are the classical conservation of flow constraints. Although the linear programming relaxation of this formulation solves the SPP, in practice, the problem is commonly solved with specialized algorithms such as Dijkstra's or Bellman-Ford (Ahuja, Magnanti & Orlin, 1993).

Several extensions of the SPP have been proposed in the literature, e.g, the Resource Constrained Shortest Path Problem (RCSPP). In this problem, the consumption of various resources is associated with each arc of the network. The objective is to find the Shortest Path (SP) connecting two nodes, while ensuring that available resource quantities are not exceeded. The RCSPP with one resource has been embedded as a pricing subproblem in column-generation procedures for vehicle routing problems (see Chabrier (2006) or Montoya, Guéret, Mendoza & Villegas (2016)).

The IP formulation of the RCSPP is obtained by adding to model (1)-(3) the following capacity (knapsack) constraints:

$$\sum_{a \in \mathcal{A}} w_a^r x_a \leq W^r, \quad \forall r \in \mathcal{R}, \tag{4}$$

where $\mathcal{R}$ is the set of constrained resources, w_a^r the consumption of resource $r \in \mathcal{R}$ by arc $a \in \mathcal{A}$, and W^r the availability of resource $r \in \mathcal{R}$.

The RCSPP belongs to the $\mathcal{NP}-$complete class (Ahuja et al., 1993) and it is usually solved by means of specialized algorithms (see, for instance, Dumitrescu & Boland (2003), Garcia (2009), Lozano & Medaglia (2013) or Horváth & Kis (2016)). The basic idea behind these algorithms is to identify and prune dominated and infeasible paths in order to reduce the size of the problem to be able to solve it via dynamic programming or similar approaches.

We address an extension of the RCSPP that, as far as we know, has not appeared in the literature: the Shared Resource Constrained Multi-Shortest Path Problem (SRC-MSPP). The problem is defined in a collection of networks $\mathcal{G}_\mathcal{N} = \{\mathcal{G}_n = (\mathcal{V}_n, \mathcal{A}_n, \mathcal{C}_n)\}_{n \in \mathcal{N}}$. For each network $\mathcal{G}_n$, the problem consists of finding the shortest path between a given source node o_n and a given sink node d_n, while not exceeding a limit in the usage of a set of common resources shared by all networks. This extension is motivated by the Resource-Constrained Multi-Project Scheduling Problems, as described in Section 4.

The motivation for the development of a heuristic solution method for the SRC-MSPP is that exact mathematical programming solvers are not able to solve instances of the size found in practice and that, as we discuss in Section 2, it is not possible to simply apply the solution methods available in the literature that have been developed for the RCSPP.

The main contributions of this work are: 1) the introduction of a of a shortest path formulation that can be employed to model Resource-Constrained Multi-Project Scheduling Problems, 2) a matheuristic search that is designed to take advantage of the modern computer architecture with multiple cores, and 3) testing and analysis of the proposed procedure to assess the contribution of its key elements.

2. Problem Description

In contrast to the RCSPP, where the arcs in a single path compete for a set of available resources, in the SRC-MSPP, the arcs in paths from multiple networks must share the constrained set of resources. Using the notation introduced in the previous section, the SRC-MSPP may be formulated as follows:

$$\min \sum_{n \in \mathcal{N}} \sum_{a \in \mathcal{A}_n} c_a x_a, \tag{5}$$

subject to:

$$\sum_{a \in \Lambda_n^+(i)} x_a - \sum_{a \in \Lambda_n^-(i)} x_a = \begin{cases} 1, & \text{if } i = o_n, \\ -1, & \text{if } i = d_n, \\ 0, & \text{otherwise.} \end{cases} \quad \forall n \in \mathcal{N}, i \in \mathcal{V}_n \tag{6}$$

$$\sum_{n \in \mathcal{N}} \sum_{a \in \mathcal{A}_n} w_a^r x_a \leq W^r, \quad \forall r \in \mathcal{R}, \tag{7}$$

$$x_a \in \{0, 1\}, \quad \forall n \in \mathcal{N}, a \in \mathcal{A}_n. \tag{8}$$

The objective function (5) minimizes the cost of the arcs in the shortest path of each network. Equations (6) are the conservation of flow constraints for each network. Constraints (7) enforce the resource limits. The model finishes with the integrality constraints (8).

The SRC-MSPP arises, for example, in the Air Traffic Flow Management (ATFM) problem associated with scheduling flight routes. The problem consists of finding optimal paths for a set of flights in such a way that no region of the airspace has more aircraft than allowed by its capacity. Likewise, no airport should be assigned more departures or arrivals than it can handle. For each aircraft, the collection of possible time-space routes can be formulated as time-expanded network (as detailed in Section 5), where each node represents a position in space, and the time instant at which that position can be reached. Thus, if no capacity restrictions existed in airports and airspace regions (constrained set of shared resources), the optimal route for each aircraft could be obtained by solving the SPP in its associated network. However, as these capacity restrictions do exist, the SPP solution is expected to be infeasible, making it necessary to solve the SRC-MSPP.

Despite the similarities between the SRC-MSPP and the RCSPP, there is a key difference between these problems. In the RCSPP, a solution (i.e., a single path) is infeasible when, for one

or more resources, it consumes more than what is available. That is, the feasibility of a path depends only on the arcs in the path and it can be detected without any additional information (local conditions). By contrast, while the path for each single network in a solution to the SRC-MSPP may be feasible (local conditions), the entire solution may not be feasible when considering the aggregated resource consumption of all paths (global conditions). In other words, instead of infeasible paths, SRC-MSPP deals with infeasible combinations of paths. This is why the methods for the RCSPP, which are largely based on identifying and pruning dominated and infeasible paths, instead of combinations of paths, are not applicable to the problem addressed here.

To illustrate this point, consider the ATFM problem described above. Under typical circumstances, all elements in a network have enough capacity to handle at least one aircraft. Therefore, no route by itself is ever infeasible (local conditions). This means that a route cannot be eliminated when considered in isolation. The feasibility of a route depends on other routes in a chosen set. Therefore, there are certain route combinations that are feasible and others that are infeasible.

3. Solution Method for the SRC-MSPP

We propose a matheuristic algorithm consisting of three phases. In the first phase, a pool $\boldsymbol{\mathcal{X}} = \{\mathcal{X}^1, \ldots, \mathcal{X}^S\}$ of solutions is generated. A solution s in the pool is represented as $\mathcal{X}^s = \{\mathcal{X}^s_n\}_{n \in \mathcal{N}}$, where $\mathcal{X}^s_n \subseteq \mathcal{A}_n$ is a set of arcs defining a path from o_n to d_n in network $\mathcal{G}_n$. This phase attempts to generate solutions $\mathcal{X}^s \in \boldsymbol{\mathcal{X}}$ that are feasible with respect to (6)-(8). However, as we discuss below, the resulting pool may include some solutions that violate one or more capacity constraints.

In the second phase, the solutions in $\boldsymbol{\mathcal{X}}$ are combined to obtain new (and perhaps better) feasible solutions, denoted by $\mathcal{X}^{\mathrm{II}}$. This is done by solving the IP model (5)-(8) with the arcs in $\boldsymbol{\mathcal{X}}$. By construction, $\mathcal{X}^{\mathrm{II}}$ cannot be worse than the best feasible solution in $\boldsymbol{\mathcal{X}}$.

The third phase applies a local search to $\mathcal{X}^{\mathrm{II}}$. The local search attempts to close the gap between the cost of each path in $\mathcal{X}^{\mathrm{II}}$ and its corresponding lower bound. The lower bound for each network can be found by solving the RCSPP. For instances of the SRC-MSPP for which all paths are feasible for an individual network, the solution of the RCSPP is equivalent to the solution of the SPP. Therefore, the lower bound for an individual network can be found by solving the SPP instead of the RCSPP. The goal of the local search is to find improved solutions, which tend to be those for which the gaps are balanced across all networks. Solution $\mathcal{X}^{\mathrm{III}}$ denotes the outcome of

the local search.

Algorithm 1 shows a pseudo-code of the proposed procedure. We define the function arguments that appear in the pseudo-code in the subsections below.

Algorithm 1 Matheuristic

1: **function** MATHEURISTIC
2: $\boldsymbol{\mathcal{X}} \leftarrow$ `GeneratePool`$(\mathcal{G}_{\mathcal{N}}, \mathcal{R}, maxIter, minIterIP, maxNets, penalty, \alpha, \beta, S)$;
3: $\mathcal{X}^{\text{II}} \leftarrow$ `SolutionCombination`$(\mathcal{G}_{\mathcal{N}}, \mathcal{R}, \boldsymbol{\mathcal{X}})$;
4: $\mathcal{X}^{\text{III}} \leftarrow$ `LocalSearch`$(\mathcal{G}_{\mathcal{N}}, \mathcal{R}, \mathcal{X}^{\text{II}}, \delta, \gamma)$;
5: **return** $\mathcal{X}^{\text{III}}$;
6: **end function**

3.1. Phase I: The `GeneratePool` Function

The goal of the first phase of our procedure is to generate a pool of feasible solutions $\boldsymbol{\mathcal{X}}$. The procedure (shown in Algorithm 2) starts with the solution of SPP[1] (i.e., model (1)–(3)) for all networks (line 2). The total cost associated with the collection of all the shortest paths, $\mathcal{X}^{\text{LB}}$, is a lower bound for the original problem. If this collection of shortest paths meets all the capacity constraints, then $\mathcal{X}^{\text{LB}}$ is an optimal solution to the original problem and the procedure terminates (lines 3–5). Otherwise (i.e., at least one capacity constraint has been violated), a for-loop to generate S feasible solutions is executed (lines 8–11). At each iteration of the loop, the `FeasibleSol` function attempts to generate a feasible solution using $\mathcal{X}^{\text{LB}}$ as a starting seed. This for-loop is amenable to parallel execution in the presence of multiple cores because the calls to `FeasibleSol` are independent. `FeasibleSol` is a non-deterministic iterative procedure that attempts to create a feasible solution $\mathcal{X}^s$ from a starting solution $\mathcal{X}^{\text{LB}}$ that does not meet the capacity constraints (5)-(8). Due to the its non-deterministic nature, it is expected that `FeasibleSol` will generate a different solution every time is called.

Algorithm 3 shows the steps associated with the function that attempts to produce a feasible solution from the collection of shortest paths $\mathcal{X}^{\text{LB}}$. The procedure identifies, for the current solution, the arcs that are contributing to the infeasibility of the solution and adds a penalty to

[1] We tackle SRC-MSPP instances for which the W^r values are such that no individual shortest path violates the capacity constraints, making the RCSPP for each network equivalent to solving the SPP. Since the networks in our computational testing are acyclic, we solve the SPP using the algorithm described in (Ahuja et al., 1993, Ch. 4.4).

Algorithm 2 Generating pool $\boldsymbol{\mathcal{X}}$

1: **function** GeneratePool($\mathcal{G}_{\mathcal{N}}, \mathcal{R}, maxIter, minIterIP, maxNets, penalty, \alpha, \beta, S$)
2: $\quad\mathcal{X}^{\text{LB}} \leftarrow$ `ShotestPath`($\mathcal{G}_{\mathcal{N}}$);
3: $\quad feasible \leftarrow$ `CheckFeasibility`($\mathcal{R}, \mathcal{X}^{\text{LB}}$);
4: $\quad$**if** $feasible ==$ `true` **then**
5: $\quad\quad\boldsymbol{\mathcal{X}} \leftarrow \{\mathcal{X}^{\text{LB}}\}$;
6: $\quad$**else**
7: $\quad\quad\boldsymbol{\mathcal{X}} \leftarrow \emptyset$;
8: $\quad\quad$**for** $s = 1, \ldots, S$ **do**
9: $\quad\quad\quad\mathcal{X}^s \leftarrow$ `FeasibleSol`($\mathcal{G}_{\mathcal{N}}, \mathcal{R}, \mathcal{X}^{\text{LB}}, maxIter, minIterIP, maxNets, penalty, \alpha, \beta$);
10: $\quad\quad\quad\boldsymbol{\mathcal{X}} \leftarrow \boldsymbol{\mathcal{X}} \cup \{\mathcal{X}^s\}$;
11: $\quad\quad$**end for**
12: $\quad$**end if**
13: $\quad$**return** $\boldsymbol{\mathcal{X}}$;
14: **end function**

the cost of a random subset of these arcs. Then, the SPP is solved for each penalized network (denoted as $\mathcal{G}_n^*$), producing shortest paths that exclude most or all of the penalized arcs. These steps are repeated in search for a feasible solution. If this fails, a final attempt to find a feasible solution is made by way of solving a reduced version of the original integer programming model. In this reduced version, some of the arcs are fixed and only a subset of the arcs is included as decision variables in the model. This step does not guarantee a feasible solution because the variable fixing may render the model infeasible. The search for a feasible solution ends once one is found or after a specified limit on the number of failed attempts. For reasons that will become clear below, adding an infeasible solution to the pool is not an issue as long as the pool contains at least one feasible solution.

The `FeasibleSol` function takes as input the set of networks (i.e., $\mathcal{G}_{\mathcal{N}}$), the set of resources ($\mathcal{R}$), the collection of shortest paths associated with the lower bound (i.e., $\mathcal{X} = \mathcal{X}^{\text{LB}}$), the maximum number of failed attempts ($maxIter$), the number of failed attempts between calls to the IP solver for the reduced model ($minIterIP$), the maximum number of networks in the reduced model ($maxNets$), the penalty value ($penalty$), the probability of penalizing a set of arcs that has been identified as contributing to the solution infeasibility (α), and the percentage of unpenalized networks that are fixed to their current paths when solving the reduced IP model (β).

Lines 2–6 initialize the local elements of the `FeasibleSol` function. Let $\mathcal{A}_n^r \subseteq \mathcal{A}_n$ be the

Algorithm 3 Producing a feasible solution from $\mathcal{X}^{\text{LB}}$

function FEASIBLESOL($\mathcal{G}_{\mathcal{N}}, \mathcal{R}, \mathcal{X}, maxIter, minIterIP, maxNets, penalty, \alpha, \beta$)
  $\mathcal{A}^{\mathcal{R}}_{\mathcal{N}} \leftarrow \{\mathcal{A}^r_n \,|\, n \in \mathcal{N}, r \in \mathcal{R}\}$;
  $\mathcal{G}^*_{\mathcal{N}} \leftarrow \mathcal{G}_{\mathcal{N}}$;
  $feasible \leftarrow$ `false`;
  $nIter \leftarrow 0$;
  $nIterIP \leftarrow minIterIP$;
  while $feasible ==$ `false` & $nIter \leq maxIter$ **do**
    $\mathcal{N}^*$, $\mathcal{G}^*_{\mathcal{N}}$, $\mathcal{A}^{\mathcal{R}}_{\mathcal{N}} \leftarrow$ `PenalizeArcs`$(\mathcal{X}, \mathcal{G}^*_{\mathcal{N}}, \mathcal{R}, \mathcal{A}^{\mathcal{R}}_{\mathcal{N}}, penalty, \alpha)$;
    if $\mathcal{N}^* == \emptyset$ **then**
      $\mathcal{A}^{\mathcal{R}}_{\mathcal{N}} \leftarrow \{\mathcal{A}^r_n \,|\, n \in \mathcal{N}, r \in \mathcal{R}\}$;
    else
      if $|\mathcal{N}^*| \leq maxNets$ & $nIterIP \geq minIterIP$ **then**
        $\mathcal{X}^{trial} \leftarrow$ `SolveIPModel`$(\mathcal{R}, \mathcal{G}_{\mathcal{N}}, \mathcal{N}^*, \mathcal{X}, \beta)$;
        $feasible \leftarrow$ `CheckFeasibility`$(\mathcal{R}, \mathcal{X}^{trial})$;
        if $feasible == true$ **then**
          $\mathcal{X} \leftarrow \mathcal{X}^{trial}$;
        else
          $nIterIP \leftarrow 0$;
        end if
      else
        $\mathcal{X} \leftarrow$ `ShotestPath`$(\mathcal{G}^*_{\mathcal{N}})$;
        $feasible \leftarrow$ `CheckFeasibility`$(\mathcal{R}, \mathcal{X})$;
      end if
    end if
    $nIterIP \leftarrow nIterIP + 1$;
    $nIter \leftarrow nIter + 1$;
  end while
  return $\mathcal{X}$;
end function

subset of arcs in network n that use resource r. We point out that an arc may belong to more than one $\mathcal{A}_n^r$. $\mathcal{A}_{\mathcal{N}}^{\mathcal{R}}$ contains the unpenalized subsets of arcs that use resource $r \in \mathcal{R}$. At the beginning, no arc subsets have been penalized and therefore all arcs subsets are included in $\mathcal{A}_{\mathcal{N}}^{\mathcal{R}}$. $\mathcal{G}_{\mathcal{N}}^* = \{(\mathcal{V}_n, \mathcal{A}_n, \mathcal{C}_n^*)\}_{n\in\mathcal{N}}$ consists of all the penalized networks and starts as a copy of $\mathcal{G}_{\mathcal{N}}$, indicating that at the beginning no arcs have been penalized (i.e., $\mathcal{C}_n^* = \mathcal{C}_n$). The $feasible$ Boolean variable keeps track of the feasibility of the solution obtained at the current iteration. The $nIter$ counter is the number of failed attempts to produce a feasible solution and $nIterIP$ counts the number of failed attempts since the last time the IP model was executed.

After the initialization, a while-loop (lines 7–27) that attempts to create a feasible solution out of the current $\mathcal{X}$ begins. In the first step of the loop, the `PenalizeArcs` function creates a list of resources for which their capacity is exceeded by the current solution. We will refer to this as the list of infeasible resources. Then, for each resource r^* in the list of infeasible resources, the procedure, with probability α, penalizes an unpenalized subset $\mathcal{A}_n^{r^*} \in \mathcal{A}_{\mathcal{N}}^{\mathcal{R}}$ if at least one arc in $\mathcal{A}_n^{r^*}$ is also in $\mathcal{X}$. The subset penalization consists of adding a *penalty* value to the current cost of all arcs in $\mathcal{A}_n^{r^*}$. That is, the penalization process is cumulative:

$$c_a^* \leftarrow c_a^* + penalty$$

Note some of the arcs in a penalized subset $\mathcal{A}_n^{r^*}$ might not be in the current solution $\mathcal{X}$. The reason for penalizing these inactive arcs is to decrease their attractiveness in subsequent iterations, since their addition to the solution may cause a resource that has been made feasible to become infeasible again. The penalization process may stop before considering all subsets that use an infeasible resource. This happens when we have penalized enough subsets that the sum of their resource requirements is at least as large as the amount by which the resource is infeasible.

`PenalizeArcs` returns the set of network indexes with at least one penalized arc in the current iteration ($\mathcal{N}^* \subseteq \mathcal{N}$), the set of networks with penalized costs ($\mathcal{G}_{\mathcal{N}}^*$), and an updated $\mathcal{A}_{\mathcal{N}}^{\mathcal{R}}$ set in which the penalized arc subsets have been removed. Note that, since an arc can consume more than one resource, removing a subset does not completely remove an arc. That is, an arc that has been penalized for consuming one resource may belong to an unpenalized subset of different resource. Since `PenalizeArcs` removes penalized arc subsets from $\mathcal{A}_{\mathcal{N}}^{\mathcal{R}}$ after each iteration, a point may be reached in which $\mathcal{A}_{\mathcal{N}}^{\mathcal{R}}$ is of a size that `PenalizeArcs` might return an empty $\mathcal{N}^*$ set. If this

occurs, $\mathcal{A}^{\mathcal{R}}_{\mathcal{N}}$ is reset to include all arcs (line 10).

As long as $\mathcal{N}^*$ is not empty, an attempt to find a feasible solution is made. The attempt takes on two different forms. An exact method solves a reduced version of the integer programming model (lines 12–20) or new shortest paths are found for the networks in $\mathcal{G}^*$ (lines 21–22). The exact method is used when the number of penalized networks is small enough (i.e.: $|\mathcal{N}^*| \leq maxNets$) and the number of attempts to reach feasibility by recomputing the shortest paths reaches $minIterIP$. The exact method solves a reduced version of the (5)–(8) model in which we fix a large percentage of the variables in the original problem. We start by selecting $\beta\%$ of the networks in $\mathcal{N} \setminus \mathcal{N}^*$. We denote this set as $\mathcal{N}^\beta$. Then, variables in the set $\{x_a \,|\, a \in \{\mathcal{A}_n\}_{n \in \mathcal{N}^\beta}\}$ are fixed to 1 if $a \in \mathcal{X}$, and to 0 otherwise. This means that the paths for the networks in $\mathcal{N}^\beta$ are fixed as dictated by the current solution. Therefore, the only variables in the reduced model are those associated with the arcs in $\{\mathcal{A}_n\}_{n \in \mathcal{N} \setminus \mathcal{N}^\beta}$. The reduced model uses the original cost values for the objective function calculation and a resource availability that is reduced by the resources requirements of the fixed variables.

In our original algorithmic design, we fixed all paths in networks without penalized arcs, i.e., all networks in $\mathcal{N} \setminus \mathcal{N}^*$. However, this proved to be too restrictive, frequently making the reduced model infeasible. The β parameter allows us to include some additional networks in the formulation and increase the flexibility of the model. The selection of the unpenalized networks to be included in the reduced model is made to balance solution quality and diversity.

After obtaining a new solution (by either of the methods described above), the procedure checks for feasibility (lines 14 and 22). If the solution is feasible, then the procedure ends and returns $\mathcal{X}$. If the solution is not feasible then the current solution changes only if the SPP method was used to find it. The current solution is not changed when the IP model is not able to find a feasible solution. The process ends after $maxIter$ failed attempts and it returns the current infeasible solution. If the solution pool does not include any feasible solutions, then `CombineSolutions` might not return a feasible solution. The probability of observing this, however, decreases with the size of the solution pool. In fact, with the size that we used in our computational experiences, the procedure never encountered this situation.

Before describing additional elements of the proposed procedure, we would like to make a brief comment on the use of the IP model to solve the reduced problem. Our original design

of the `FeasibleSol` function did not include this component. We added it after preliminary computational experiences showed that, for small $|\mathcal{N}^*|$ values, feasibility could be reached faster and with better solution quality by solving the reduced problem instead of continuing to penalize arcs and finding the revised shortest paths. In this design, the IP exact solver is meant to be invoked occasionally. That is, it is not meant to be the first option. Therefore, the values of the $maxNets$ and $minIterIP$ parameters must be chosen accordingly. The $maxNets$ parameter controls the size of the subproblem. Given that we are using an exact solver, we need to limit the size of the model that we are asking the solver to tackle. The $minIterIP$ parameter is a proxy for directly monitoring the changes in $\mathcal{N}^*$. Note that if the IP model fails to produce a feasible solution with a particular $\mathcal{N}^*$, it only makes sense to invoke it again after $\mathcal{N}^*$ has experienced some changes. Instead of keeping track of the changes in $\mathcal{N}^*$, we experimentally adjusted the value of $minIterIP$ to allow the arc penalization function to change the composition of $\mathcal{N}^*$.

3.2. Phase II: The `SolutionCombination` Function

As discussed at the beginning of Section 3, the `SolutionCombination` function combines the solutions in $\boldsymbol{\mathcal{X}}$ and produces a new solution $\mathcal{X}^{\text{II}}$, as long as at least one solution in the pool is feasible. The combination process consists of solving a reduced IP model (5)–(8), where the only variables are those associated with arcs in the pool of solutions. That is, the set of variables in the model is $\{x_a\}_{a \in \boldsymbol{\mathcal{X}}}$.

This form of combination of solutions has two key properties. First, the resulting solution $\mathcal{X}^{\text{II}}$ is at least as good as the best feasible in $\boldsymbol{\mathcal{X}}$. In our computational experiments, we observed that $\mathcal{X}^{\text{II}}$ was always better than any of the solutions in $\boldsymbol{\mathcal{X}}$. Second, $\mathcal{X}^{\text{II}}$ is guaranteed to be no worse than any solution found as a combination of the paths associated with the solutions in $\boldsymbol{\mathcal{X}}$. This is due to the generation of $\mathcal{X}^{\text{II}}$ by a combination of arcs (instead of paths) that allows forks and joints to be produced at the nodes of the subnetwork induced by the arcs in $\boldsymbol{\mathcal{X}}$, leading to paths that are not in the pool of solutions. Our experiments with various designs led us to conclude that this combination method is superior to others in terms of the quality of the combined solution and the computational time to find it.

3.3. Phase III: The `LocalSearch` Function

In preliminary experimentation we observed that, for a given solution $\mathcal{X}^{\text{II}}$ resulting from the combination method, some networks used paths whose cost was much higher (relative to the known

lower bound) than the cost associated with other networks. For each network n, we calculate this difference as follows:

$$\Delta_n = \sum_{a \in \mathcal{A}_n} c_a (x_a^{\mathrm{II}} - x_a^{\mathrm{LB}}) \tag{9}$$

We concluded that the $\mathcal{X}^{\mathrm{II}}$ solutions tend to be unbalanced, with a relatively small number of networks with much larger Δ_n values than others. We therefore developed the `LocalSearch` function taking into account the structure of the $\mathcal{X}^{\mathrm{II}}$ solutions. In particular, `LocalSearch` focuses on improving the paths in networks with relatively large Δ_n values. This is done at the possible expense of worsening the delta values of other networks. The search, in other words, is for a balanced solution. That is, one for which the collection of delta values for all networks have less variance.

The local search uses two parameters, δ and γ to operate on $\mathcal{X}^{\mathrm{II}}$, where δ represents the number of networks with the largest Δ_n values and γ is the number of networks sharing at least one resource with the networks with the largest Δ_n values.

The procedure starts by identifying the set of δ networks with the largest Δ_n values. This set ($\mathcal{N}^{\delta}$) contains the networks for which the local search is trying to improve their Δ_n values. The procedure then selects, from all the networks not in $\mathcal{N}^{\delta}$, γ networks ($\mathcal{N}^{\gamma}$), each of them sharing at least one resource with one or more networks in $\mathcal{N}^{\delta}$. The networks in $\mathcal{N}^{\gamma}$ are used as "partners" for the networks in $\mathcal{N}^{\delta}$ in order to trade off the use of resources. The values of the parameters associated with the local search are such that $\delta \ll \gamma$.

We define $\mathcal{N}^{\mathrm{LS}} = \mathcal{N}^{\delta} \cup \mathcal{N}^{\gamma}$ and solve (5)–(8) by fixing the paths in $\mathcal{N} \setminus \mathcal{N}^{\mathrm{LS}}$. That is, variables in the set $\{x_a \,|\, a \in \{\mathcal{A}_n\}_{n \in \mathcal{N} \setminus \mathcal{N}^{\mathrm{LS}}}\}$ are fixed to 1 if $a \in \mathcal{X}^{\mathrm{II}}$, and to 0 otherwise; and the remaining variables (i.e., $\{x_a \,|\, a \in \{\mathcal{A}_n\}_{n \in \mathcal{N}^{\mathrm{LS}}}$) are the only ones in the IP model. The solution of the IP model is denoted by $\mathcal{X}^{\mathrm{III}}$. This solution is guaranteed to be no worse than $\mathcal{X}^{\mathrm{II}}$. We have observed that the local search, as defined above, is often able to improve upon the solution constructed by the combination method, except in those cases when $\mathcal{X}^{\mathrm{II}}$ is near-optimal. We have experimented with a local search that focuses only on the reduced set of networks with the worst Δ_n values (i.e., the networks in $\mathcal{N}^{\delta}$) and determined that this strategy provides very little room for improvement because most of the resources are committed to the paths that are fixed prior to solving the IP model.

4. Resource-Constrained Project Scheduling Problem

The SRC-MSPP can be applied to Resource-Constrained Project Scheduling Problems (RCPSP). In its simplest version (see Kolisch & Padman (2001), Kolisch & Hartmann (2006) or Hartmann & Briskorn (2010) or Zheng & Wang (2015)), the RCPSP consists of finding a schedule that minimizes the project's completion time (i.e., the makespan), where a project is characterized by a set of activities with precedence relationships and resource availability. A feasible schedule of activities is such that it does not violate the precedence constraints and it does not consume more than the available resources.

Several extensions of the basic RCPSP have been proposed in literature. For example, (Confessore, Giordani & Rismondo, 2007), (Gonçalves, Mendes & Resende, 2008) or (Krüger & Scholl, 2009) address the problem with multiple projects and common resources needs.

The multi-mode extension occurs when several executions modes are considered, i.e., when the execution mode determines the resource requirements and completion times. Examples of this extension can be found in (Kuster, Jannach & Friedrich, 2009) or (Kellenbrink & Helber, 2015), where the authors deal with an even more elaborate multi-mode scheme in which the activities required to complete a project might change (i.e., they are not fixed) and that the choice of modes is independent for each activity.

Extensions related to when activities can be executed have also been developed. This has been studied considering time windows in which resources are not available (e.g., during holidays), as in (Lu, Ren, Wang & Zhu, 2019) and considering forbidden time periods in which activities cannot be executed. As discussed by (Hartmann & Briskorn, 2010), in contrast to time windows where the availability of resources affect all activities, forbidden periods are particular to each activity.

A common assumption in the aforementioned publications is that all the projects must be scheduled. However, sometimes, due to resource limitations (e.g., budget), not all the projects can be executed and a subset must be selected. This situation, described for instance in (Chen & Askin, 2009), gives rise to the extension of the RCPSP with project selection.

The basic characteristics of the project scheduling problems that can be modeled with the SRC-MSPP are:

1. Multiple projects sharing resources whose availability might vary during the planning horizon.

2. Each project has a starting time window and activities can only be scheduled within a given time interval.
3. There is no idle time between activities, i.e., once an activity has been completed, the following activity must start immediately.

We note that there are workarounds in order to apply SRC-MSPP to a scheduling problem that does not conform with the last characteristic (i.e., one that considers idle times between activities). The following additional characteristics may also be considered:

1. Setup times required for the use of some resources.
2. Multi-modes associated with individual activities.
3. Project selection.
4. Flexibility in the type of objective function, e.g., minimize the makespan or the sum of the cost of executing each activity on a given mode ((Achuthan & Hardjawidjaja, 2001)).

We now discuss how the SRC-MSPP is capable of capturing the various characteristics of the RCPSP and its extensions.

4.1. Resource-Constrained Multi-Project Scheduling Problems solved as SRC-MSPP

Let $n \in \mathcal{N}$ be a project consisting of a set of activities $\mathcal{A}_n$ that must be executed in a particular order (i.e, the execution of the activities must obey a set of precedence relationships). Let $\mathcal{A}_\mathcal{N}$ represent the set of all activities. Project n is completed once the activities in $\mathcal{A}_n$ have been completed. Projects must be completed within a planning horizon represented as a set of discrete time periods. Each project must start within a time window and the duration of each activity has a minimum, a scheduled, and a maximum number of periods. Activities consume capacity from a set of resources $\mathcal{R}$ (e.g., machines). For each time period that activity $a \in \mathcal{A}_\mathcal{N}$ is being processed, the activity consumes an amount w_a^r of the capacity of resource r. Preemption of projects and activities is not allowed, i.e., once a project or activity has started, it must be completed. Machines have no waiting queue[2]. Precedence relationships between projects might exist, such that, for a given sequence of projects, a project cannot start until the previous one has been fully completed.

[2]Note that queues, and therefore idle times between activities, can be modeled as dummy machines.

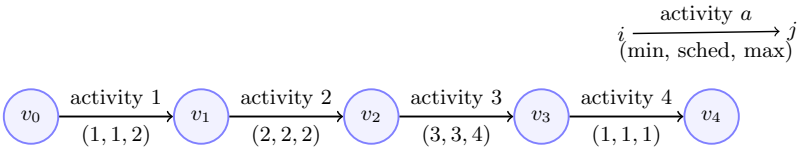

Figure 1: Example of a sequence of activities for a project.

The objective is to find a feasible schedule for all the activities included in the projects, so they are completed at minimum cost while respecting machines' capacity restrictions. Therefore, the problem is to determine the starting processing time of each project and the processing speed of the machines. For each activity, the chosen machine speed determines the activity processing time.

Since each project is made of a sequence of activities, it can be represented as a directed graph where arcs correspond to activities and vertices represent start and completion events. Figure 1 shows an example of a 4-activity project represented as an activity-on-arc graph, where vertex v_a represents the completion of activity a and the three values below each arc represent the number of time periods for the minimum, the scheduled, and the maximum duration of the activity. In this example, the four activities are scheduled to be completed in 1, 2, 3, and 1 time periods.

The start time for each activity in a project can be derived from the possible start times of the project and the possible duration of the activity. For instance, assume that possible start times for the project in Figure 1 are $\{1, 2, 3\}$, then the possible start times for each activity are:

$$\{1, 2, 3\}, \{2, 3, 4, 5\}, \{4, \ldots, 7\}, \{7, \ldots, 11\},$$

and the possible completion times for the project are $\{8, \ldots, 12\}$.

An extended graph is built by expanding each vertex in the original graph by the possible start times of each activity. For instance, activity 1 in Figure 1 has three possible start times (i.e., 1, 2, and 3). Vertex v_0 represents the start of activity 1. Therefore, the extended graph will have three vertices representing the three possible start times for this activity (see vertices labeled (v_0, 1), (v_0, 2), and (v_0, 3) in Figure 2). This is done for all the vertices in the original network. Arcs in the extended graph represent the possible activity duration. For instance, the arc from vertex (v_0, 2) to vertex (v_1, 4) represents the case when activity 1 starts in period 2 and has a duration of two periods.

The extended graph has an origin (o) and a destination (d) vertex. The origin vertex is linked

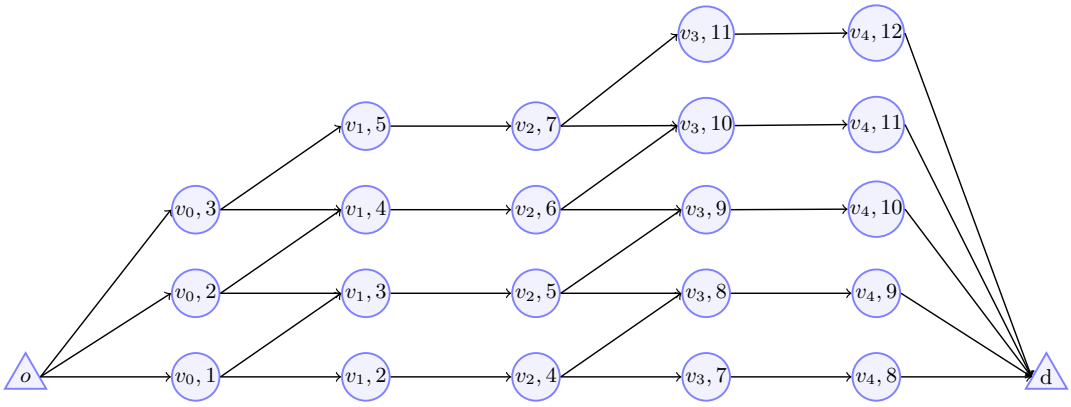

Figure 2: Extended graph for the project in Figure 1

to every tail vertex of the first activity and the destination vertex is linked to every head vertex of the last activity, as shown in Figure 2.

A feasible schedule for a project corresponds to a path from the origin to the destination in its associated extended graph. A value can be assigned to each arc of the extended graph in such a way that it represents the cost of starting the activity in the time period indicated in its tail vertex and finishing the activity in the time period indicated in its head vertex. The resource unconstrained problem of finding the optimal schedule of activities is equivalent to finding the shortest path in the extended graph. However, since in most settings, projects share resources, the constrained scheduling problem for all projects can be solved as an instance of the SRC-MSPP.

4.2. Multi-modes and Project Selection

The multi-mode extension of the RCPSP may be modeled by allowing a sequence of activities to be replaced by an alternative sequence. This is equivalent to changing the assignment of some activities to a different machine or replacing a subset of activities. For instance, suppose that in the project depicted in Figure 1, activities 2 and 3 can be substituted with a new sequence of activities 5-6, such that, if this new sequence is selected, activity 5 cannot be processed until activity 1 is finished and activity 4 must start immediately after activity 6 is completed. Figure 3 represents this situation and Figure 4 depicts the corresponding extended graph.

It is also possible to include project selection within the framework of solving these scheduling problems by means of finding paths in extended graphs. To illustrate this, let us consider that we have a subset of projects such that exactly one of the projects must be selected. This new situation

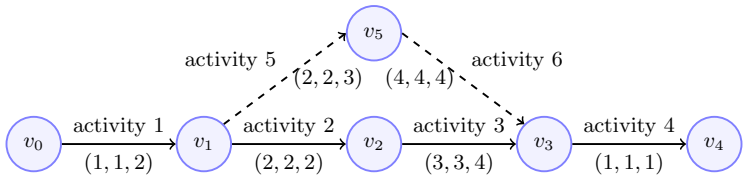

Figure 3: Example of alternative sequences of activities for a project.

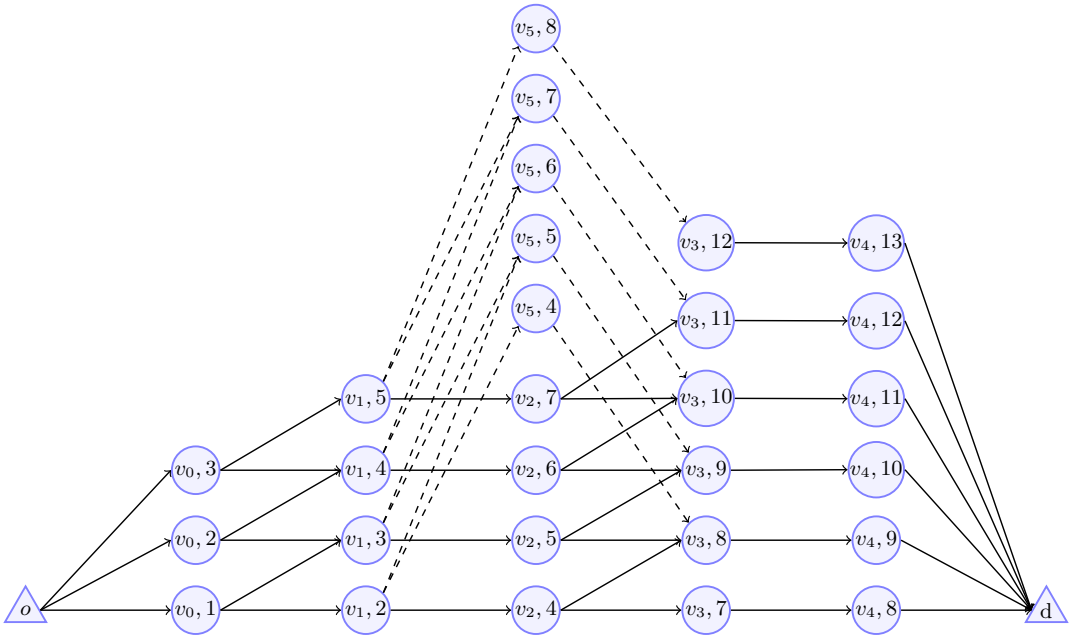

Figure 4: Extended graph for the project in Figure 3.

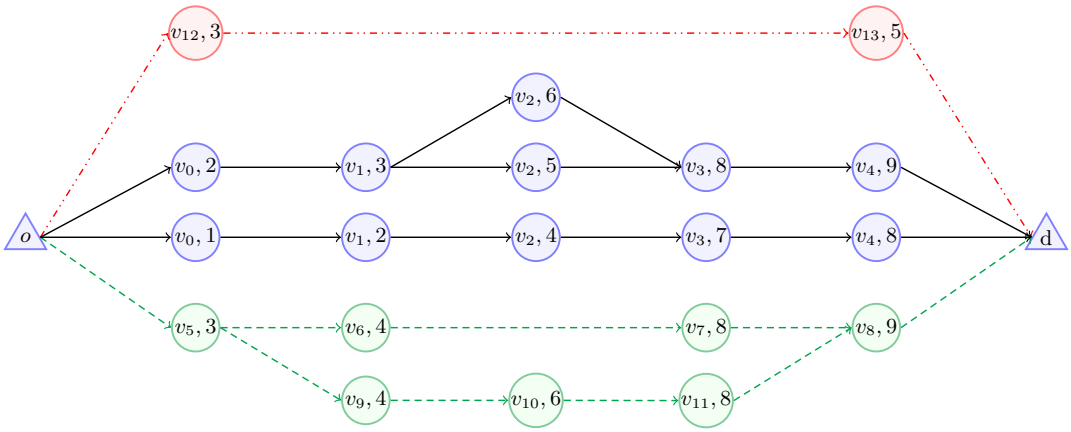

Figure 5: Project selection and scheduling.

can be represented in a graph where each selected and scheduled project corresponds to a path from an origin to a destination. The graph for project selection consists of merging all the extended graphs for the candidate projects. The origins and destinations are merged so that only one origin and one destination exist in the extended graph for project selection. The case that allows for no project to be selected can be represented by adding a dummy project with a single activity to be processed on a machine with unlimited capacity. Each project selection (including the option of a null selection) and scheduling corresponds to a path from an origin to a destination.

Figure 5 shows the situation in which at most one of two projects must be selected, one depicted with solid lines, and the other with dashed lines. In addition to these two projects, the graph includes a dummy project (dotted-dash lines), representing the choice of not selecting either of the two real projects.

Assigning to each non-dummy arc an operation cost and assigning to the dummy arc the cost of not selecting any of the projects, makes the problem of optimally selecting and scheduling at most one project equivalent to the Shortest Path Problem from o to d. For the problem in which the entire set of projects is partitioned in mutually exclusive subsets from which only one project could be selected, the optimal project selection and scheduling corresponds to the solution of the SRC-MSPP.

5. Computational Experience

We test our procedure on a set of instances of an Air Traffic Flow Management (ATFM) problem, which represents a context where the SRC-MSPP arises in practice. In the ATFM problem (see a detailed description in García-Heredia, Alonso-Ayuso & Molina (2019)), a plan for a set of flights is developed months in advanced to determine the path that each flight must follow from departure to destination. The path specifies the airports and air traffic control points (waypoints) that a flight must cross, as well as the scheduled time to pass through each of these control points. The time horizon of the flight plan is discretized in time periods (e.g., 5 minutes). Each aircraft passes through several disjoint partitions of the airspace called sectors. Sectors, as well as airports, have limited capacity, i.e., security restrictions limit the number of aircraft that are allowed to fly through a sector in each time period. The capacity of the fly sectors and the airports may change due to issues such as weather, rendering the original flight plans infeasible. That is, in a weather event, executing original flight plans might violate the revised capacities associated with fly sectors or airports. In other situations (e.g., in busy air traffic days), the original plans may simply exceed the capacity of some sectors or airports for a few periods of time. Either way, the operator of the ATFM system (central decision-maker) must propose some changes to the original plans within a few hours prior to a flight while minimizing the costs produced by the proposed changes. The modifications applied to flight plans may include alternative routes, ground delays or/and air delays/advances in the arrivals at some waypoints/airports. An interesting feature of this problem is the existence of continued flights, i.e., flights that must wait to take off until a previous one has landed, usually because both are operated by the same aircraft. This causes a situation in which a decision for one flight may propagate to the entire network of flights.

The ATFM problem has appeared in the literature (see for instance Bertsimas & Patterson (1998), Bertsimas, Lulli & Odoni (2011) or Agustín, Alonso-Ayuso, Escudero & Pizarro (2012)), but only once has been formulated as a SRC-MSPP (García-Heredia et al., 2019). In this approach, each aircraft corresponds to a network where the potential modifications to the flight plan are the arcs. Air sectors and airports are the set of resources with limited capacity. For ATFM, the SRC-MSPP has weights w_a^r equal to 1 in constraints (7) because the capacity W^r is given as number of aircraft.

5.1. Problem Instances

We use publicly available flight plans for our computational experience (Bureau of Transportation Statistics (a,b)). The data correspond to domestic flights[3] in the US for the 16th of January, May, and September of 2019. The choice of the 16th is due to the high-volume of air traffic on that day for each of these months.

The flight plans provided us with data such as departure and landing times. We set the maximum ground delay for each flight to 90 minutes and the maximum air delay/increase to 25% of the flight travel time. This time information defines the minimum, maximum, and scheduled times for each flight.

As typically done in the ATFM literature, we generated the data for the airspace configuration, that is, the waypoints and the capacity limits for sectors and airports. We divided the airspace into 400 sectors using a 20×20 grid. There are nine waypoints in each sector (see Figure 6), eight on the boundary, to define entry and exit points; and one in the middle, to connect the boundary-waypoints by bi-directional arcs (solid lines in the figure). Each airport is also connected

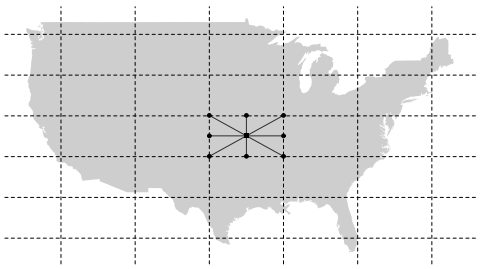

Figure 6: Example of sectors and waypoints.

to its sector boundary-waypoints by bi-directional arcs. Thus, we use the graph resulting from the waypoints and arcs to define aircraft routes, where the main route is the shortest path between the departure and landing airports. The arcs included in a route represent activities to be completed within a time window. Main routes along with the times in the flight plans enable us to compute the minimum capacity required at airports and sectors to avoid changes in the flight plans (see

[3]Flights outside the contiguous territory of the US (e.g.: Alaska), and those marked as canceled or diverted (landing at a destination other than the originally scheduled) were not considered.

example in Figure 7). We employ this information to assign alternative routes (multi-modes of the projects) and capacity limits to airports and sectors.

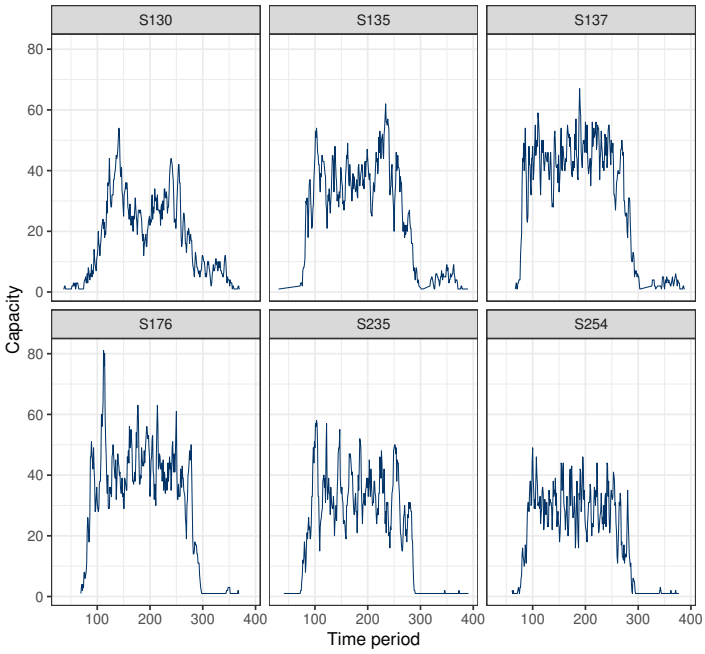

Figure 7: Capacity requirements per time period for the most demanded sectors on January 16, 2019.

We included alternative routes for flights whose main route passed through at least one of the twenty sectors with the largest capacity requirements. These new routes are the shortest path between the departure and landing airport that do not go through some of the twenty most demanded sectors. Several alternative routes can be generated depending on which high-demand sectors are allowed to be used. We generated a maximum of four alternative routes for each flight.

We set capacity limits for airports and sectors via penalization methods that test the robustness of the proposed solution method. For each flight plan, we tested twelve scenarios, which we grouped in three categories (easy, medium, and difficult). We first created a base scenario of capacity values which is based on the minimum value required for the original flight plan to be feasible. In addition, the base scenario sets the limit on the number of simultaneous departures and landings occurring at an airport as 80% of its total departure and arrival capacity.

Easy cases simulate the effect of bad weather moving across sectors and causing capacity reductions (Bertsimas et al., 2011). Concretely, the generation of these instances starts with one of the five most demanded sectors (chosen at random) and two contiguous sectors. The capacity of

these three sectors (and their airports) is reduced during five consecutive time periods. Then, the capacity reduction is moved to three other contiguous sectors for another five time periods. The process is repeated for the entire time horizon. The reductions were set at 10%, 20%, 30% and 40% of the base capacity, leading to a total of four scenarios. Medium cases are based on the easy ones with an additional capacity reduction randomly applied to 50% of the elements and enforced during the entire planning horizon. The additional reduction of the capacity of each element is randomly chosen between 1% and 20%. For the difficult cases, the randomly generated capacity reduction is applied to all elements. A minimum capacity of one aircraft was set for each element in all the scenarios.

In order to build a test set with various problem sizes, we created smaller versions of the original flight plans with 30% and 65% of the total number of aircraft[4]. The dimensions of the instances in our test set are shown in Table 1. Note that each arc corresponds to a variable in the IP model (5)-

Table 1: Dimensions of the instances in our test set.

Flight plan	Size	#Flights	#Networks	#Arcs	#Nodes	#Resources
Jan	30%	5,596	1,329	3,116,931	1,482,732	136,887
May	30%	6,951	1,368	4,043,046	1,868,888	145,711
Sep	30%	6,610	1,368	3,6706,44	1,729,221	142,734
Jan	65%	11,788	2,879	7,195,904	3,336,779	173,249
May	65%	13,752	2,963	9,717,333	4,341,805	178,451
Sep	65%	13,530	2,964	8,597,363	3,922,250	185,519
Jan	100%	18,100	4,429	11,934,419	5,446,911	184,404
May	100%	20,634	4,558	14,475,487	6,457,815	204,724
Sep	100%	20,581	4,559	13,993,972	6,248,302	201,451

(8), each node to a flow constraint, and each resource to a capacity constraint. The large number of capacity constraints is due to the product of the number of capacity elements and periods in the planning horizon. There are twelve scenarios for each case in the table, resulting in a total of 108 instances. Since most of the constraints in the problem define facets (flow constraints), a strong LP relaxation is expected when attempting to solve the problem using exact methods. Our computational experiments corroborate this important characteristic of the model, which others

[4]The generation of capacity levels is flight plan-depended, so they had to be repeated for these smaller plans.

have pointed out as related to time-expanded network IP formulations (Boland & Savelsbergh, 2019).

5.2. Parameter Setting

We perform all experiments on a HP Z230 Tower Workstation, with 4 cores (processor i7-4770 3.40GHz) and 32 GB of RAM. Our MIP solver is Gurobi 9.0 (Gurobi Optimization (2020)). The algorithm was coded in C++, compiled using the GNU compiler 6.3, and run on a Debian 9 Operating System. The for-loop to generate the pool of solutions (Algorithm 2) was parallelized using OpenMP (OpenMP Architecture Review Board (2015)) with 8 threads. The code and data sets can be found in `https://github.com/DavidGarHeredia/SRC-MSPP`.

To take into account the random elements in the proposed procedure, we solved each problem instance five times. We represent the associated variability with violin plots, a well-known extension of the boxplot that shows the distribution of the data. In violin plots, the quantiles corresponding to 25%, 50% (median), and 75% are depicted with a solid line, while a dark diamond shows the mean.

We used ParamILS (Hutter, Hoos & Stützle, 2007) to set the values for the parameters of our search procedure, except for the value of S, the size of the solution pool. (Below, we discuss how we set this value.) ParamILS is an automated system for parameter setting and algorithm configuration. The goal of this system is to optimize a target algorithm's performance on a given class of problem instances by varying a set of ordinal and/or categorical parameters. We measured performance as a function of quality and solution time. We made slight adjustments to the settings found by ParamILS based on our knowledge of our search procedure. These are the parameter values that we used for our experimentation:

- $maxIter = 100$
- $minIterIP = 20$
- $maxNets = 5\%|\mathcal{N}|$
- $penalty = 6000$ (about four times the maximum cost)
- $\alpha = 50\%$

- $\beta = 95\%$
- $\delta = 2\%$
- $\gamma = 30\%$

In addition, we set the MIP gap in Gurobi at various levels depending on the purpose for calling this optimizer: solving full model (0.5%), generating the pool of solutions (1%), combining the solutions (1%), and applying local search (0.5%).

5.3. Integer Programming Results

We attempted to solve the IP model for the 108 instances in our test set with Gurobi. However, 44 of the runs terminated in an out-of-memory error and without an integer solution. These instances correspond to the full flight plans and for the 65% reduced flight plans for the May and September days under the difficult scenario. Table 2 summarizes the results of the Gurobi runs.

Table 2: Summary of Gurobi results.

Flight plan	Size	Difficulty	$\bar{t}$	t_Δ	$root_\%$	$\overline{Gap}$	Gap_Δ	$col_\%$	$row_\%$
Jan	30%	easy	126.84	1.12	75%	0.08%	[0, 0.34]%	17.81%	38.34%
Jan	30%	medium	238.34	1.87	0%	0.11%	[0.05, 0.15]%	17.94%	38.39%
Jan	30%	difficult	337.61	1.19	0%	0.34%	[0.27, 0.4]%	18.12%	38.62%
May	30%	easy	172.56	1.15	100%	0%	[0, 0]%	16.32%	35.77%
May	30%	medium	253.61	1.68	75%	0.02%	[0, 0.06]%	16.39%	35.80%
May	30%	difficult	722.91	2.33	0%	0.33%	[0.18, 0.44]%	16.50%	35.89%
Sep	30%	easy	155.68	1.14	100%	0%	[0, 0]%	17.38%	37.69%
Sep	30%	medium	163.59	1.16	100%	0%	[0, 0]%	17.40%	37.65%
Sep	30%	difficult	505.21	1.41	0%	0.44%	[0.35, 0.52]%	17.51%	37.70%
Jan	65%	easy	335.97	1.38	75%	0.04%	[0, 0.15]%	16.61%	36.13%
Jan	65%	medium	856.33	1.79	25%	0.14%	[0, 0.24]%	16.68%	36.18%
Jan	65%	difficult	2,117.18	1.44	0%	0.29%	[0.25, 0.39]%	16.75%	36.28%
May	65%	easy	523.56	1.72	75%	0.05%	[0, 0.21]%	14.87%	33.35%
May	65%	medium	1,271.99	1.89	25%	0.17%	[0, 0.37]%	14.91%	33.39%
Sep	65%	easy	374.24	1.05	100%	0%	[0, 0]%	15.73%	34.74%
Sep	65%	medium	862.39	1.30	0%	0.31%	[0.29, 0.38]%	15.78%	34.76%

The first three columns of the table show the date of the flight plan, the percentage of flights chosen from the flight plan (Size), and the level of difficulty. This is followed by the average solution

time in seconds ($\bar{t}$) and the $\frac{t_{\max}}{t_{\min}}$ ratio (t_Δ), where $t_{\max}$ and $t_{\min}$ are the worst and best solution times. t_Δ measures solution time variability, where values close to one indicate less sensitivity to the different weather scenarios. $root_\%$ denotes the percentage of instances solved at the root node of the Branch & Bound (B&B) tree. In all other instances, only an additional node had to be explored. $\overline{Gap}$ and Gap_Δ are the average and the range of the integrality gap, defined as $100\frac{z^{\text{IP}}-z^{\text{LP}}}{z^{\text{LP}}}$, where z^{LP} and z^{IP} are the objective function value of the LP relaxation and the integer solution, respectively. These values along with with $root_\%$ measure the strength of the IP formulation. $col_\%$ and $row_\%$ represent the percentage of columns and rows that Gurobi deleted in the preprocessing phase.

The following observations relate to the values in Table 2:

1. Gurobi's solution times are within a range that is acceptable in practice for the ATFM problem (35 minutes in the worst case).
2. The value of t_Δ is greater than or equal to 1.3 in 10 out of 16 cases, indicating that for a given difficulty level, weather scenarios have a significant impact (i.e., 30% difference) in the solution time.
3. As customary in B&B, solution times exhibit nonlinear increases with the problem size.
4. $root_\%$, $\overline{Gap}$ and Gap_Δ show that the formulation has a strong LP relaxation, with 18 out of 64 instances solved at the root node and an integrality gap of less than 0.45% after one node.
5. The last two columns show that Gurobi achieved a significant reduction of the original size of the problem.

The ATFM problems have a very specific cost structure, as described in (García-Heredia et al., 2019). To test the sensitivity of Gurobi to the cost structure in the IP formulation, we generated costs from a continuous uniform distribution $(0, 100)$ for the arcs in the networks in our problem test set. This new cost structure turned out to increase the difficulty of the problems in such a way that, out the 108 instances, Gurobi was able to solve only 24. All of these instances belong to the sets of size 30%.

5.4. Adjusting Pool Size

Before assessing the performance of our procedure, we experimented with the size of the solution pool. Recall that we did not use ParamILS to adjust this parameter. We used the instances of size

30% to test four different pool size values (16, 32, 48, and 64). The results are summarized in Figure 8 (a) and (b), where violin plots show, for each difficulty level and pool size, the distribution of the optimaltiy gap and the speed-up with respect to Gurobi's solutions. Given the $m-$th instance and the $k-$th execution of our heuristic procedure on that instance, the optimality gap is computed as $100\frac{z^{\mathrm{h}}_{m,k}-z^{\mathrm{IP}}_{m}}{z^{\mathrm{IP}}_{m}}$, where $z^{\mathrm{h}}_{m,k}$ and z^{IP}_{m} denote the objective function values for the solutions found with our heuristic and with Gurobi, respectively. Similarly, the speed-up is computed as $\frac{t^{\mathrm{IP}}_{m}}{t^{\mathrm{h}}_{m,k}}$, where $t^{\mathrm{h}}_{m,k}$ and t^{IP}_{m} denote the wall time required by our procedure and the one required by Gurobi, respectively.

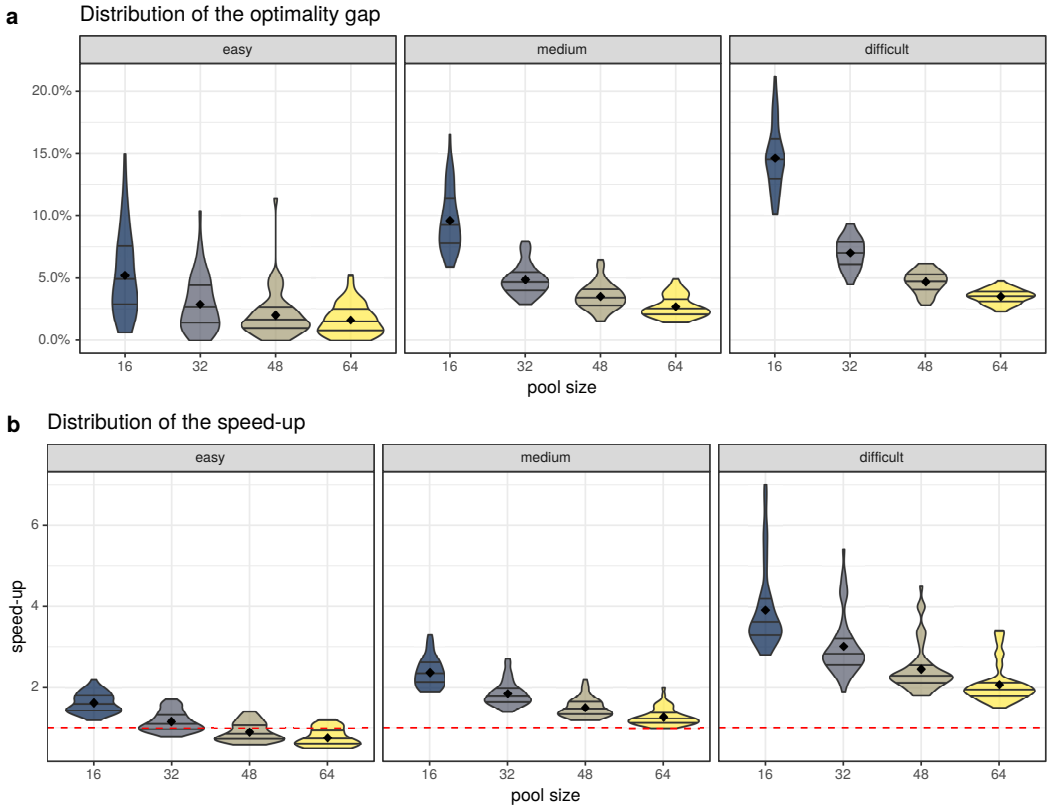

Figure 8: Effect of the pool size for instances of size 30%.

Figure 8 (a) shows the importance of the pool size as the difficulty of the problem increases. Larger pool sizes achieve smaller optimality gaps with lower variability. For instance, for a pool size of 64, the optimality gap is always less than 5%.

The dashed line in Figure 8 (b) shows the speed-up value of 1, which is when the heuristic solution time is the same as Gurobi's. Values below this line mean that the optimal solution was

found and confirmed faster than the running time of the heuristic. This happened in some of the easy cases, particularly, for the two largest pool sizes. The heuristic finished before Gurobi in the medium and difficult instances. The run-time difference is more significant in difficult cases, where the minimum speed-up is 1.5.

In terms of computational effort, we observed that our parallelization scheme for generating the pool of solutions scaled linearly. That is, twice as much time is required to generate the same amount of solutions with half the threads. Extrapolating from the experiments with instances of size 30% and in order to balance solution quality and computational effort, we chose a pool of 80 solutions for instances of size 65% and a pool of 96 solutions for instances with the complete flight plans.

5.5. Performance Assessment

We used the smallest problem instances to find the best values for our search parameters as well as adjusting the size of the solution pool. We now assess the performance of the procedure with the larger problem instances. As shown in Table 2, Gurobi was able to solve seven of the nine cases for instances of size of 65% of the original flight plans, namely the three sets for January and the "easy" and "medium" sets for May and September. Therefore, we are able to calculate optimally gaps and speed-up values when applying our procedure to these problem instances. Figure 9 (a) and (b) summarize the results.

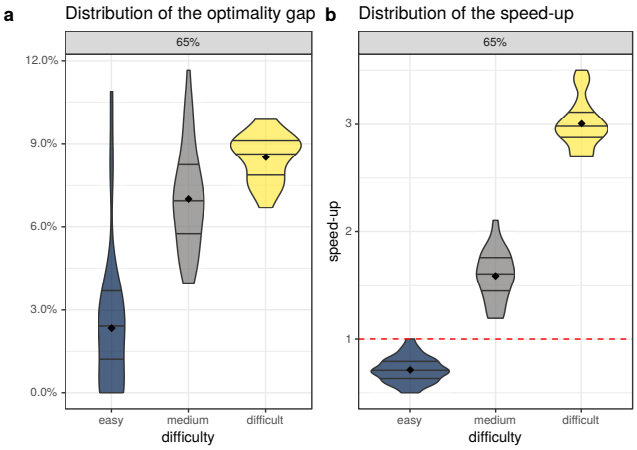

Figure 9: Results of the algorithm for instances of size 65% that exact methods could also solve.

The first figure shows that the proposed heuristic generates high-quality solutions for these

instances. For the easy cases, the heuristic solutions are, on average, less than 3% away from optimality. For the medium cases, in more than 75% of the times the optimality gap is below 9% and the average is less than 7.5%. For the difficult cases, the optimality gap is below 10%, with an average of 8.5%. In terms of computational effort, our speed-up calculation shows that the heuristic was faster than the exact method in the medium and difficult cases, but not in the easy ones. It is interesting to point out that in the instances when the proposed heuristic is faster than the exact method, Gurobi is still solving the LP relaxation at the root node when the heuristic has already finished.

A possible conclusion from these results is that Gurobi should be the preferred method to solve the easy problems, since it can confirm optimality before our procedure finishes. However, this result depends on the cost structure. Recall that when we used randomly generated costs, Gurobi was only able to solve 24 out of the 108 test problems. A comparison of optimality gap and speed-up between our procedure and Gurobi for those 24 instances is shown in Figure 10 (a) and (b). We observe that, under this cost structure, the solutions achieved by our procedure are near-optimal, and they are reached faster than Gurobi (with a minimum speed-up of 1.5).

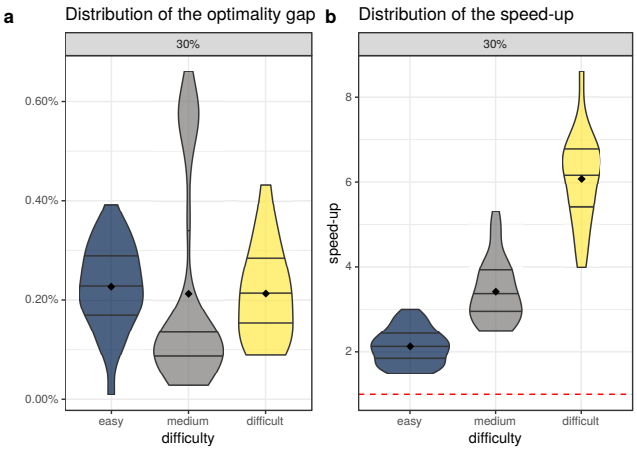

Figure 10: Results using a random cost structure.

This experiment shows that the performance of the exact method within Gurobi is sensitive to the cost structure. On the other hand, our heuristic-based approach exhibits a robust performance across cost structures. Furthermore, we observed that the solutions that we found after the combination method have an optimality gap of 1%. Therefore, in case where computational time is limited, it would seem sensible to skip the local search phase when tackling problems with this

cost structure.

For the flight plans of May and September under the difficult scenario in the 65% set, and all the instances in the 100% set, Gurobi, due to memory limitations, was unable to even solve the LP relaxation of the problem. Therefore, we were not able to assess the quality of our solutions with the lower bound corresponding to the LP relaxation. We attempted to find solutions for these instances with LocalSolver (LocalSolver (2020)), a general-purpose optimizer based on metaheuristic principles and methodologies, but this software also failed to handle them. In order to obtain lower bounds, we dualized the capacity constraints of problem (5)-(8) and solved the corresponding Lagrangian Relaxation. Since the non-dualized constraints (flow constraints) define facets for the problem, the optimum of the Lagrangian dual problem is guaranteed to be equal to the LP relaxation of problem (5)-(8) (Guignard (2003)). Based on the gaps reported in Table 2, the lower bounds resulting from the Lagrangian Relaxation are expected to be tight.

Figure 11 shows the gap between all the solutions found throughout our experimentation and the Lagrangian bound. The dashed line is the mean gap of the heuristic solution with respect to the Lagrangian bound. The band around the mean, represents the range of gaps obtained from the

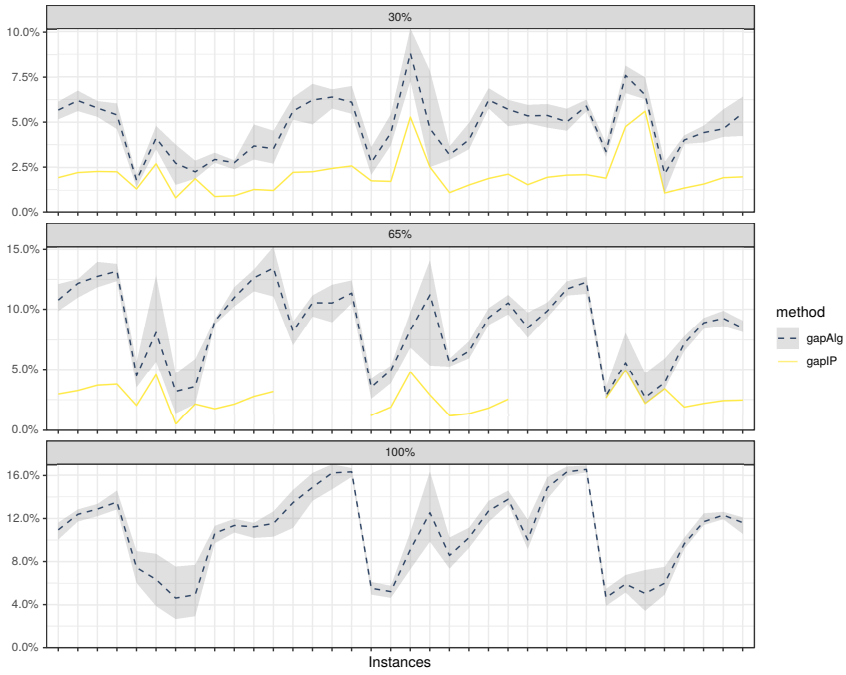

Figure 11: Deviation from Lagrangian lower bounds.

5 runs of our procedure. Similarly, the solid line in the figure shows the gap of Gurobi's solutions

(when available) with the Lagrangian bound.

The general observation from Figure 11 is that the deviation from the Lagrangian bounds increases with the size of the problem. This is true for both the heuristic and the optimal solutions. For the full flight plans, our procedure obtained, in the worst case, solutions with average gaps of 16%. However, in most cases the gap varied between 4% and 14.5%. Considering that these gaps are against a lower bound, these results are more than reasonable for the practical application that we studied. Note that some of the optimal solutions have gaps of up to 5% with respect to the lower bound. In terms of computational time, our procedure found the solutions to most of the full-size problems in less than 25 minutes. In the worst case, the procedure took 40 minutes. These computational times are within the valid range for ATFM.

5.6. Analysis of the Search Procedure

The preceding results show that we are able to produce high-quality solutions in reasonable computational times. Our procedure was able to find solutions to problems that a commercial solver could not handle. The purpose of the following analysis is to provide insights on the performance of our procedure.

Figure 12 shows the average percentage of time that our procedure employs in each phase, as a function of the problem size and the level of difficulty. We observe that: 1) the solution pool

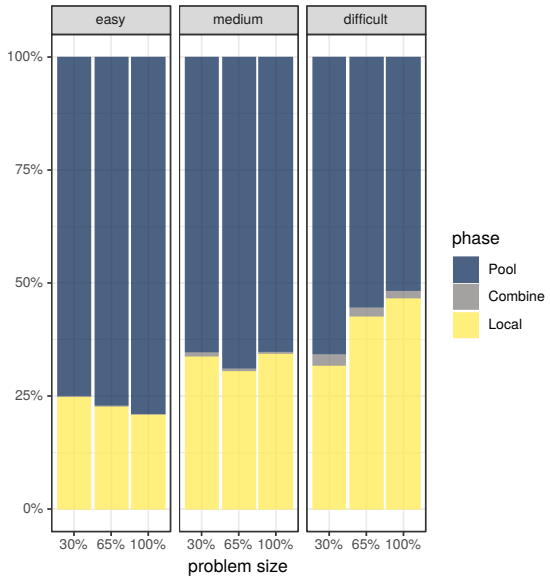

Figure 12: Percentage of solution time in each phase of the proposed solution procedure.

generation represents the bottleneck, consuming more than 50% of the solution time; 2) combining solutions consumes a negligible amount of time; and 3) the time required by local search is sensitive to the difficulty of the problem.

The good news of the solution pool generation being the procedure's bottleneck is that, as mentioned before, the parallelization of this phase linearly scales with the number of threads. This means that when generating a pool of 96 solutions, if instead of 8 threads we had 48, the time in this phase would have been $\frac{1}{6}$ of the time that we report. In such a situation, the procedure could be set up to generate more solutions, which in turn could improve the quality of the results.

Figure 13 (a) and (b) shows the percentage of improvement in solution quality achieved after each phase of the procedure, as a function of the problem size and level of difficulty. Figure 13

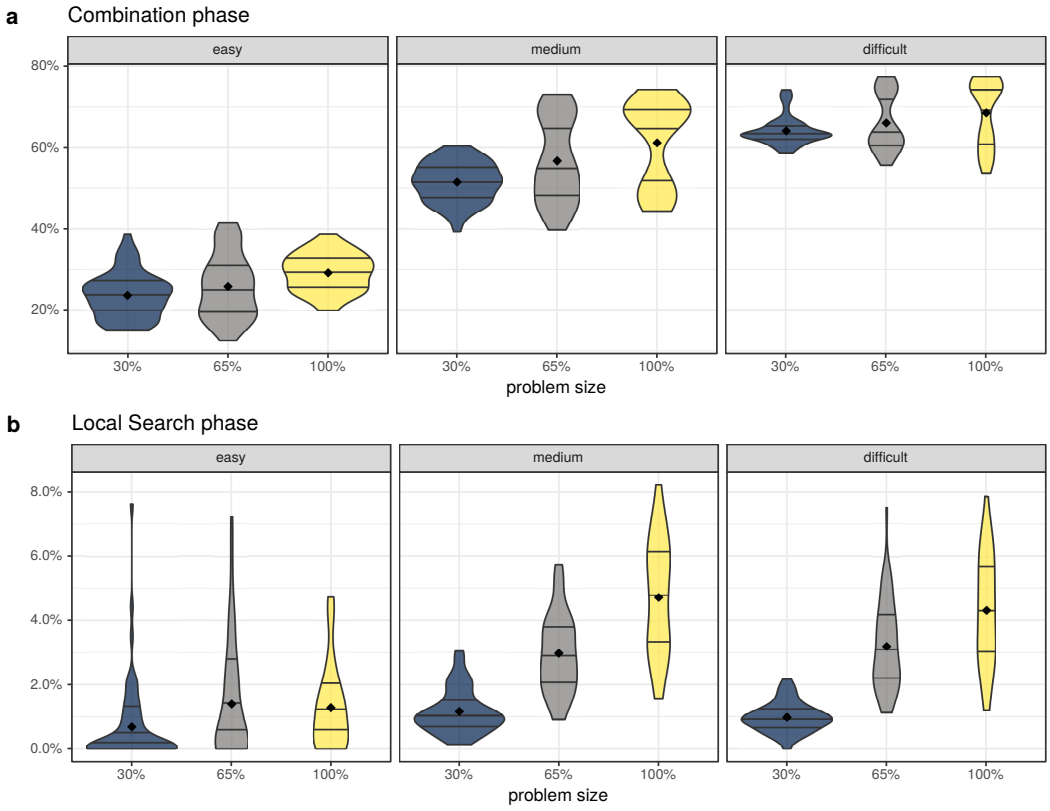

Figure 13: Improvement achieved after each phase of the proposed procedure.

(a) reveals the large improvement achieved by the solution combination phase, particularly as the difficulty of the problem increases. The large solution-quality improvement combined with the modest amount of computational time make this phase one of the key elements for the success of

the solution method.

When compared to the solution combination phase, the local search achieves a significantly smaller improvement in solution quality. This is mainly due to the reduced opportunities for improvement after solutions are combined. Nonetheless, the improvement achieved by the local search justifies its computational requirements and therefore its inclusion as part of the solution process.

To conclude the analysis of the proposed procedure, we point out that we designed the algorithmic elements to take advantage of the structure of the problem. In particular, the generation of the solution pool exploits a very specific characteristic of our problem, which turns out to also be true in the context of the RCSPP. In general, the least expensive arcs are also those that consume the most limited resources at a higher rate. The key is to generate a set of diverse solutions that include the use of arcs with limited or nonexistent consumption of the most limited resources. Typically, these arcs are the most costly or else the trivial solution in which all origins and destinations are connected by their shortest paths would be feasible and therefore optimal. The controlled random element included in the generation of solutions results in a diversity of arcs in the solution pool that is then exploited by the combination method

6. Conclusions and Future Research

In this work we presented a matheuristic algorithm for the Shared Resource Constrained Multi-Shortest Path Problem (SRC-MSPP). As it has been shown, the SRC-MSPP can be used to solve Resource-Constrained Multi-Project Scheduling Problems. A particular case where these problems arise is in Air Traffic Flow Management (ATFM), a problem that has been used to test the algorithm. Insights on how and why the algorithm works have been exposed along the computational experience. Among the conclusions drawn for the algorithm, we emphasize the following: 1) It is parallel-computing oriented, with a design that allows to efficiently take advantage of all the cores available in a computer, 2) It achieves good-quality solutions in short periods of time, 3) It can solve bigger and more difficult instances than exact methods, and 4) It is robust across changes problem size, difficulty, and cost structure.

Future research directions include: 1) Exploring a local search phase able to achieve larger improvements in shorter time, and 2) Addressing the stochastic version of the problem. A stochastic

approach to the problem would be particularly relevant in the context of ATFM.

References

Achuthan, N., & Hardjawidjaja, A. (2001). Project scheduling under time dependent costs–a branch and bound algorithm. *Annals of Operations Research*, *108*, 55–74.

Agustín, A., Alonso-Ayuso, A., Escudero, L. F., & Pizarro, C. (2012). On air traffic flow management with rerouting. part I: Deterministic case. *European Journal of Operational Research*, *219*, 156–166.

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows: theory, algorithms, and applications*. Upper Saddle River (New Jersey): Prentice Hall.

Bertsimas, D., Lulli, G., & Odoni, A. (2011). An integer optimization approach to large-scale air traffic flow management. *Operations research*, *59*, 211–227.

Bertsimas, D., & Patterson, S. S. (1998). The air traffic flow management problem with enroute capacities. *Operations research*, *46*, 406–422.

Boland, N. L., & Savelsbergh, M. W. (2019). Perspectives on integer programming for time-dependent models. *Top*, *27*, 147–173.

Chabrier, A. (2006). Vehicle routing problem with elementary shortest path based column generation. *Computers & Operations Research*, *33*, 2972–2990.

Chen, J., & Askin, R. G. (2009). Project selection, scheduling and resource allocation with time dependent returns. *European Journal of Operational Research*, *193*, 23–34.

Confessore, G., Giordani, S., & Rismondo, S. (2007). A market-based multi-agent system model for decentralized multi-project scheduling. *Annals of Operations Research*, *150*, 115–135.

Dumitrescu, I., & Boland, N. (2003). Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks: An International Journal*, *42*, 135–153.

Garcia, R. (2009). *Resource constrained shortest paths and extensions*. Ph.D. thesis Georgia Institute of Technology.

García-Heredia, D., Alonso-Ayuso, A., & Molina, E. (2019). A combinatorial model to optimize air traffic flow management problems. *Computers & Operations Research*, *112*, 104768.

Gonçalves, J. F., Mendes, J. J., & Resende, M. G. (2008). A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research*, *189*, 1171–1190.

Guignard, M. (2003). Lagrangean relaxation. *Top*, *11*, 151–200.

Gurobi Optimization, L. (2020). Gurobi optimizer reference manual. URL: `http://www.gurobi.com` accessed on September 3, 2020.

Hartmann, S., & Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research*, *207*, 1–14.

Horváth, M., & Kis, T. (2016). Solving resource constrained shortest path problems with lp-based methods. *Computers & Operations Research*, *73*, 150–164.

Hutter, F., Hoos, H. H., & Stützle, T. (2007). Automatic algorithm configuration based on local search. In *Aaai* (pp. 1152–1157). volume 7.

Kellenbrink, C., & Helber, S. (2015). Scheduling resource-constrained projects with a flexible project structure. *European Journal of Operational Research*, *246*, 379–391.

Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European journal of operational research*, *174*, 23–37.

Kolisch, R., & Padman, R. (2001). An integrated survey of deterministic project scheduling. *Omega*, *29*, 249–272.

Krüger, D., & Scholl, A. (2009). A heuristic solution framework for the resource constrained (multi-) project scheduling problem with sequence-dependent transfer times. *European Journal of Operational Research*, *197*, 492–508.

Kuster, J., Jannach, D., & Friedrich, G. (2009). Extending the rcpsp for modeling and solving disruption management problems. *Applied Intelligence*, *31*, 234.

LocalSolver (2020). Localsolver reference manual. URL: `https://www.localsolver.com/` accessed on September 3, 2020.

Lozano, L., & Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. *Computers & Operations Research*, *40*, 378–384.

Lu, Z., Ren, Y., Wang, L., & Zhu, H. (2019). A resource investment problem based on project splitting with time windows for aircraft moving assembly line. *Computers & Industrial Engineering*, .

Montoya, A., Guéret, C., Mendoza, J. E., & Villegas, J. G. (2016). A multi-space sampling heuristic for the green vehicle routing problem. *Transportation Research Part C: Emerging Technologies*, *70*, 113–128.

OpenMP Architecture Review Board (2015). OpenMP application program interface version 4.5. URL: `https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf` accessed on September 3, 2020.

Bureau of Transportation Statistics, U. D. o. T. (a). Airline on-time performance data. URL: `https://www.transtats.bts.gov/tables.asp?db_id=120&DB_Name=` accessed on September 3, 2020.

Bureau of Transportation Statistics, U. D. o. T. (b). Aviation support tables. URL: `https://www.transtats.bts.gov/tables.asp?DB_ID=595&DB_Name=&DB_Short_Name=` accessed on September 3, 2020.

Zheng, X.-l., & Wang, L. (2015). A multi-agent optimization algorithm for resource constrained project scheduling problem. *Expert Systems with Applications*, *42*, 6039–6049.